Universidade do Minho
Escola de Engenharia

Departamento de Informática

Licenciatura em Engenharia Informática

Laboratórios de Informática III

Projeto de Laboratórios de Informática III – 2.ª Fase

Grupo n.º 24

Alexandre Rodrigues Fernandes, n.º A94154

Orlando José da Cunha Palmeira, n.º A97755

Rui Pedro Guise da Silva, n.º A97133

Braga, dezembro de 2021

Índice

1 Introdução

Nesta segunda fase do projeto, foi-nos proposto que o nosso programa desse resposta a um conjunto de dez questões (doravante designadas por *queries*) sobre os dados provenientes de três ficheiros que contêm informações sobre commits, repositórios e utilizadores.

As quatro primeiras *queries* dão informações sobre o panorama estatístico dos dados, tais como as quantidades dos tipos de utilizadores (se são *bots*, utilizadores ou organizações), a quantidade média de colaboradores por repositório, número de *bots* colaboradores e o número médio de commits por utilizador.

As restantes *queries* (parametrizáveis) são interativas e são dependentes de parâmetros inseridos pelo utilizador. Podemos dizer que o *output* da maioria destas *queries* são uma espécie de *ranking* sobre os dados. Algumas informações que podemos obter são, por exemplo, os utilizadores mais ativos num certo período de tempo, repositórios que perderam a sua atividade após uma certa data, as linguagens de programação mais utilizadas num certo intervalo de tempo, entre outras.

2 Queries Estatísticas

A obtenção dos dados necessários para a resolução das *queries* estatísticas foi efetuada simultaneamente com a criação dos catálogos dos utilizadores, repositórios e commits.

2.1 Querie 1 – Quantidade de bots, utilizadores e organizações

A recolha dos dados para a resolução desta querie foi obtida em tempo de parsing do ficheiro dos utilizadores.

Ao analisar cada linha do ficheiro, avaliámos o parâmetro “type” e conforme o valor nele presente, incrementámos a variável respetiva que armazena a quantidade do tipo.

2.2 Querie 2 – Número médio de colaboradores por repositório

A recolha dos dados para a resolução desta querie foi obtida em tempo de parsing do ficheiro dos commits.

Ao analisar cada linha do ficheiro, procurámos o utilizador que fez o commit e caso esse utilizador ainda não tenha commits associados, incrementámos o valor da variável que conta o número de colaboradores.

O número de repositórios necessário para calcular o resultado desta querie é obtido durante o parsing do ficheiro dos repositórios.

2.3 Querie 3 – Número de repositórios contendo bots como colaboradores

A recolha dos dados para a resolução desta querie foi obtida em tempo de parsing do ficheiro dos commits.

Ao analisar cada linha do ficheiro, procurámos o utilizador que fez o commit e verificámos se o mesmo é um bot. Se tal acontecer, incrementámos o valor da variável que armazena a quantidade de bots colaboradores.

2.4 Querie 4 – Quantidade média de commits por utilizador

A recolha dos dados para a resolução desta querie foi obtida em tempo de parsing do ficheiro dos commits e dos utilizadores.

Após o parsing dos ficheiros acima referidos, obtemos o número total de commits e o número total de utilizadores (dado pela soma users + bots + organizações, evitando a criação de uma outra variável que armazena o número total de utilizadores).

3 Queries Parametrizáveis

3.1 Querie 5 – Top N de utilizadores com mais commits num intervalo de datas

Para a resolução desta *querie*, utilizámos uma função que percorre o catálogo dos utilizadores e, em cada utilizador, vai efetuar uma verificação aos seus commits.

Ao acedermos aos commits do utilizador, percorremo-los e avaliámos a data em que os mesmos foram efetuados. Se a data de um dado *commit* desse utilizador estiver entre as datas fornecidas pelo ficheiro dos comandos, então o *commit* é contabilizado nesse utilizador.

No final de todos os cálculos, temos um *buffer* (lista ligada) com o resultado dessa querie.

O output no ficheiro é feito através da travessia desse *buffer* em que recolhemos os dados nele contidos.

3.2 Querie 6 – Top N de utilizadores com mais commits em repositórios de uma dada linguagem

Para a resolução desta *querie*, utilizámos uma função que percorre o catálogo dos utilizadores e, em cada utilizador, vai efetuar uma verificação aos seus commits.

Ao acedermos aos commits do utilizador, percorremo-los e procurámos o id do repositório onde cada commit foi feito. Posteriormente, verificámos se a linguagem associada a esse repositório é igual à fornecida nos parâmetros. Se for igual, esse commit é contabilizado nesse utilizador.

No final de todos os cálculos, temos um *buffer* (lista ligada) com o resultado dessa querie.

O output no ficheiro é feito através da travessia desse *buffer* em que recolhemos os dados nele contidos.

3.3 Querie 7 – Lista de repositórios sem commits a partir de uma determinada

Para a resolução desta *querie*, utilizámos uma função que percorre o catálogo dos utilizadores e, em cada utilizador, vai efetuar uma verificação aos seus commits.

Nos commits desse utilizador, recolhemos o commit com a data mais recente e verificámos se a mesma é anterior à data fornecida nos parâmetros. Se tal acontecer, o repositório associado a esse commit é considerado inativo a partir dessa data. No entanto, se futuramente se encontrar um utilizador que tenha feito um commit nesse repositório posterior à data fornecida, o repositório deixa de ser considerado inativo.

No final de todos os cálculos, temos um *buffer* (lista ligada) com o resultado dessa querie.

O output no ficheiro é feito através da travessia desse *buffer* em que recolhemos os dados nele contidos.

Esta querie tem algumas limitações, uma vez que a obtenção dos seus resultados é um processo pesado que tem uma influência negativa na performance do programa, nomeadamente no seu tempo de execução.

É provável que o atraso desta querie seja devido a uma procura constante dos repositórios no *buffer* dos resultados para, eventualmente, alterar o seu estado de inativo para ativo.

3.4 Querie 8 – Top N de linguagens mais utilizadas a partir de uma determinada data.

Para a resolução desta *querie*, utilizámos uma função que percorre o catálogo dos repositórios e, em cada um, vai recolher a sua data de criação e a linguagem de programação.

Se a data de criação do repositório for mais recente (ou igual) à data fornecida nos parâmetros da *querie,* a linguagem de programação desse repositório é contabilizada.

No final de todos os cálculos, temos um *buffer* (lista ligada) com o resultado dessa querie.

O output no ficheiro é feito através da travessia desse *buffer* em que recolhemos os dados nele contidos.

3.5 Querie 9 – Top N de utilizadores com mais commits em repositórios cujo owner seu amigo.

Para a resolução desta *querie*, utilizámos uma função que percorre o catálogo dos utilizadores e, em cada utilizador, vai efetuar uma verificação aos seus commits.

Em cada commit, recolhemos o seu *repo_id* (id do repositório do commit) e vemos se o *owner_id* desse repositório pertence à lista de seguidores do utilizador verificado e também à lista de quem esse utilizador segue. Se tal acontecer, esse utilizador é contabilizado com o seu número de commits.

No final de todos os cálculos, temos um *buffer* (lista ligada) com o resultado dessa querie.

O output no ficheiro é feito através da travessia desse *buffer* em que recolhemos os dados nele contidos.

3.6 Querie 10 – Top N de utilizadores com as maiores mensagens de commit por repositório.

Para a resolução desta *querie*, utilizámos uma função que percorre o catálogo dos utilizadores e, em cada utilizador, vai recolher o tamanho da maior mensagem feita por ele nalgum commit.

À medida que percorremos os utilizadores, vamos inserindo no *buffer* dos resultados os utilizadores e o tamanho da sua maior mensagem. No fim da execução da *querie,* temos o *buffer* com os N utilizadores que fizeram as maiores mensagens de commit.

No entanto, a *querie* não apresenta os resultados desejados visto que devolve o N utilizadores com maiores mensagens de commit em vez de ser os N utilizadores com as maiores mensagens por repositório.

4 Preocupações tidas no projeto

4.1 Gestão de Memória

Para um melhor aproveitamento do uso da memória RAM no armazenamento dos dados, decidimos ter em conta os seguintes aspetos:

1. Uma vez que sabemos que ID´s e datas são sempre valores positivos, optámos por utilizar o tipo de dados *unsigned int* em vez do usual *int,* pois este último gasta 1 bit para a representação do sinal do número. Deste modo, conseguimos armazenar maiores gamas de valores numéricos.
2. Para a identificação de cada tipo de utilizador, atribuímos-lhe um número correspondente (1 para bots, 2 para users e 3 para organizações). Uma vez que os valores utilizados para esta identificação são muito pequenos, optámos por utilizar o tipo de dados *char* uma vez que este ocupa apenas 1 byte evitando, assim, o uso do *int* que ocupa 4 bytes.
3. Usando uma abordagem semelhante ao ponto 2, no fator de balanceamento das árvores binárias também utilizámos o tipo de dados *char* pois os únicos valores que são utilizados no balanceamento são -1, 0 e 1. Se o fator de balanceamento for igual a -1, fazemos uma rotação à direita, se for igual a 1 fazemos uma rotação à esquerda.

4.2 Performance

Para haver uma maior eficiência nas pesquisas feitas nos catálogos, optámos por armazená-los em árvores binárias de procura balanceadas.

4.3 Prevenção de erros e debug

Para evitar a ocorrência de erros na utilização da memória, tais como, a não libertação quando já não é necessária e leituras e escritas em endereços indevidos, utilizámos a ferramenta Valgrind para fazer esses testes ao nosso programa.

```
gcc -g -O2 -Wno-unused-result -ansi -Wall -Wextra -pedantic -std=gnu99 ./src/bstrees.c ./src/fileUtils.c ./src/repos.c ./src/co
mmits.c ./src/users.c ./src/linkedLists.c ./src/readWrite.c ./src/parsing.c ./src/main.c -o guiao-2 -lm
valgrind -s --leak-check=full ./guiao-2 commands2.txt
==3137== Memcheck, a memory error detector
==3137== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3137== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3137== Command: ./guiao-2 commands2.txt
==3137==
==3137==
==3137== HEAP SUMMARY:
==3137==     in use at exit: 0 bytes in 0 blocks
==3137==   total heap usage: 4,262,768 allocs, 4,262,768 frees, 134,468,421 bytes allocated
==3137==
==3137== All heap blocks were freed -- no leaks are possible
==3137==
==3137== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figura 1 - Output do Valgrind

5 Custos da implementação utilizada

5.1 Memória

Tal como indicado na Figura 1, o uso total de memória RAM utilizada para esta implementação (com o set de ficheiros fornecidos para o guião 2) foi de 134 468 421 bytes (aproximadamente 128,24 MB).

Como consequência de guardar todos os dados na memória, o desempenho de algumas das queries foi penalizado, nomeadamente a querie 7, uma vez que os commits ficam separados por cada utilizador. Porém, aceitámos os custos associados a esta querie uma vez que beneficia o desempenho das queries 5, 6 e 9.